Companie Aeriana - Proiect TPJAD

Schiopu Adrian, Anca Iulia, Iacob Alexandru

December 16, 2025

1 Introducere

Aplicatia dezvoltata este o platforma de cumparare si rezervare a biletelor de avion. Aceasta are rolul de a ajuta utilizatorii sa gaseasca rapid zboruri disponibile si sa faca rezervari intr-un mod simplu si eficient.

Problema pe care aplicatia web o rezolva este gestionarea zborurilor si a rezervarilor intr-un singur sistem. Utilizatorii pot cauta zboruri si pot rezerva bilete direct din aplicatie. Astfel, se reduce timpul necesar pentru cautarea informatiilor si realizarea unei rezervari.

Scopul proiectului este dezvoltarea unei aplicatii usor de utilizat, care sa ofere functionalitati clare atat pentru utilizatori, cat si pentru administratori. Utilizatorii pot vizualiza si rezerva zboruri, iar administratorii pot adauga, modifica sau sterge zboruri din sistem.

2 Arhitectura sistemului

Aplicatia este dezvoltata sub forma unei aplicatii web de tip client–server. Aceasta arhitectura permite separarea clara intre partea de interfata utilizator si partea de procesare a datelor si a logicii aplicatiei.

Partea de client este realizata folosind React, fiind responsabila de interfata grafica si interactiunea cu utilizatorul. Clientul permite cautarea zborurilor, realizarea rezervarilor si gestionarea actiunilor disponibile in functie de tipul de utilizator (utilizator normal sau administrator).

Partea de server este implementata folosind Java Spring, avand rolul de a gestiona logica aplicatiei, validarea datelor si comunicarea cu baza de date. Serverul expune API-uri REST care sunt apelate de client pentru operatii precum autentificare, afisare zboruri, rezervari si administrarea zborurilor.

Aplicatia foloseste H2 Database ca baza de date relationala, in care sunt stocate informatiile despre utilizatori, zboruri si rezervari. Gestionarea dependintelor si procesul de build sunt realizate cu ajutorul Maven.

2.1 Componentele principale ale sistemului

Client web (React)
Interfata grafica a aplicatiei, responsabila pentru afisarea datelor si preluarea actiunilor utilizatorului.

Server backend (Spring)
Contine logica aplicatiei, controalele pentru autentificare, gestionarea zborurilor si a rezervarilor, precum si accesul la baza de date.

Baza de date (H2 Database)
Stocheaza datele aplicatiei in trei tabele principale:

- USER – contine informatii despre utilizatori
- FLIGHT – contine informatii despre zboruri
- RESERVATION – realizeaza legatura intre utilizatori si zboruri

2.2 Structura bazei de date

Tabelul USER contine urmatoarele campuri:

- id (int)
- username (string)
- password (string)
- isAdmin (boolean)

Tabelul FLIGHT contine urmatoarele campuri:

- id (int)
- departure (string)
- arrival (string)
- departure_time (LocalDateTime)
- duration (int)
- nrSeats (int)
- plane_name (string)

Tabelul RESERVATION contine urmatoarele campuri:

- id (int)
- userID (int)
- flightID (int)

Relatia dintre tabele este una de tip *one-to-many*, unde un utilizator poate avea mai multe rezervari, iar un zbor poate fi rezervat de mai multi utilizatori. Tabelul RESERVATION face legatura intre USER si FLIGHT.

2.3 Diagrama bazei de date

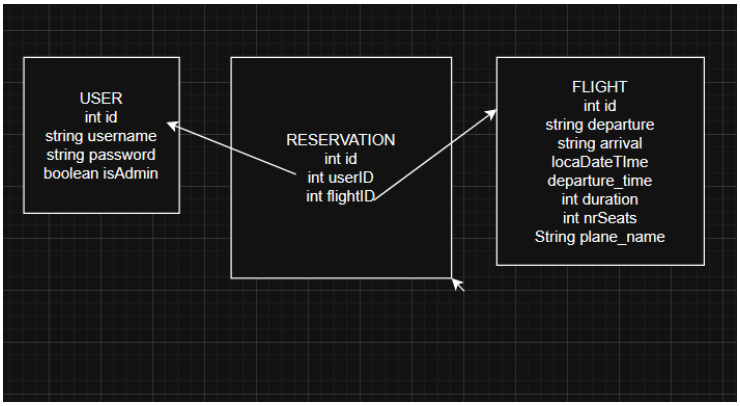

Figure 1: Diagrama relatiilor dintre tabelele bazei de date

3 Structura proiectului

3.1 Implementarea backend-ului (Spring)

Backend-ul aplicatiei este organizat pe straturi, respectand arhitectura clasica *Model -Controller - Service - Repository*. Fiecare strat are un rol bine definit, asigurand separarea logicii aplicatiei de accesul la date si de expunerea serviciilor REST.

3.2 Repository layer

Stratul de tip *repository* este responsabil pentru interactiunea directa cu baza de date. Acesta foloseste Spring Data JPA pentru realizarea operatiilor CRUD (Create, Read, Update, Delete).

3.2.1 FlightRepository

FlightRepository extinde interfata *JpaRepository*, oferind operatii standard asupra entitatii Flight. In plus, acesta contine metoda *findByDepartureAndArrival*, care permite cautarea unui zbor in functie de aeroportul de plecare si de sosire. Aceasta metoda este utilizata pentru functionalitatea de cautare a zborurilor.

3.2.2 ReservationRepository

ReservationRepository gestioneaza accesul la datele rezervarilor si ofera metode pentru:

- cautarea rezervarilor asociate unui utilizator

- cautarea rezervarilor pentru un anumit zbor
- verificarea existentei unei rezervari pentru o combinatie utilizator–zbor

Acest repository realizeaza legatura dintre utilizatori si zboruri prin intermediul tabelei RESERVATION.

3.2.3 UserRepository

UserRepository este responsabil pentru gestionarea datelor utilizatorilor. Acesta permite cautarea unui utilizator dupa username si este utilizat in procesul de autentificare al aplicatiei.

3.3 Service layer

Stratul *service* contine logica aplicatiei si intermediaza comunicarea dintre controllere si repository-uri.

3.3.1 FlightService

FlightService gestioneaza operatiile asupra zborurilor si ofera metode pentru:

- obtinerea tuturor zborurilor
- cautarea unui zbor dupa ID
- salvarea si stergerea zborurilor
- cautarea zborurilor dupa aeroportul de plecare si sosire

3.3.2 ReservationService

ReservationService gestioneaza rezervarile si permite:

- crearea unei rezervari
- stergerea unei rezervari
- obtinerea rezervarilor pentru un utilizator
- obtinerea rezervarilor pentru un zbor

3.3.3 UserService

UserService gestioneaza operatiile asupra utilizatorilor si permite:

- inregistrarea utilizatorilor
- cautarea utilizatorilor
- stergerea conturilor
- cautarea dupa username pentru autentificare

3.4 Modelele aplicatiei

Modelele reprezinta entitatile principale ale aplicatiei si corespund tabelelor din baza de date.

3.4.1 Modelul User

Modelul User reprezinta un utilizator al aplicatiei. Acesta contine informatii precum identificatorul unic, username-ul, parola si rolul utilizatorului (administrator sau utilizator normal).

3.4.2 Modelul Flight

Modelul Flight reprezinta un zbor disponibil in sistem. Acesta contine informatii despre aeroportul de plecare, aeroportul de sosire, data si ora plecarii, durata zborului, numarul de locuri disponibile si numele avionului.

3.4.3 Modelul Reservation

Modelul Reservation reprezinta o rezervare efectuata de un utilizator pentru un zbor. Acesta realizeaza legatura dintre utilizator si zbor, continand referinte catre identificatorul utilizatorului si al zborului.

3.5 Controller layer (API REST)

Controller-ele expun functionalitatile backend-ului sub forma de *API-uri REST*, care sunt apelate de aplicatia frontend.

3.5.1 FlightController

FlightController expune endpoint-uri pentru:

- listarea tuturor zborurilor
- obtinerea unui zbor dupa ID
- adaugarea unui zbor nou
- modificarea unui zbor existent
- stergerea unui zbor
- cautarea zborurilor dupa plecare si sosire

Operatiile de creare, modificare si stergere sunt destinate administratorilor.

3.5.2 ReservationController

ReservationController gestioneaza rezervarile si permite:

- afisarea tuturor rezervarilor
- crearea unei rezervari
- modificarea unei rezervari
- stergerea unei rezervari
- obtinerea rezervarilor unui utilizator
- obtinerea rezervarilor pentru un zbor

3.5.3 UserController

UserController gestioneaza utilizatorii si permite:

- crearea unui cont nou
- modificarea datelor unui utilizator
- stergerea unui utilizator
- autentificarea utilizatorilor prin username si parola

Metoda de autentificare verifica existenta utilizatorului si corectitudinea parolei si returneaza datele utilizatorului daca autentificarea este valida.

3.5.4 Exemplu de utilizare a endpoint-urilor

Toate endpoint-urile aplicatiei sunt accesibile folosind urmatorul URL de baza:

`http://localhost:8080/api`

3.5.5 User Endpoints

Get All Users
`GET /api/users`
Raspuns: `200 OK`

```
[
  {
    "id": 1,
    "username": "admin",
    "admin": true
  }
]
```

Get User by ID
`GET /api/users/{id}`
Raspuns: `200 OK | 404 Not Found`
Create User
`POST /api/users`
Content-Type: application/json

```
{
  "username": "newuser",
  "password": "password123",
  "admin": false
}
```

Raspuns: `201 Created`
Update User
`PUT /api/users/{id}`
Raspuns: `200 OK | 404 Not Found`
Delete User
`DELETE /api/users/{id}`
Raspuns: `204 No Content | 404 Not Found`
Login
`POST /api/users/login`
Raspuns: `200 OK | 401 Unauthorized`

3.5.6 Flight Endpoints

Get All Flights
`GET /api/flights`
Raspuns: `200 OK`
Get Flight by ID
`GET /api/flights/{id}`
Raspuns: `200 OK | 404 Not Found`
Create Flight
`POST /api/flights`
Raspuns: `201 Created`
Update Flight
`PUT /api/flights/{id}`
Raspuns: `200 OK | 404 Not Found`
Delete Flight
`DELETE /api/flights/{id}`
Raspuns: `204 No Content | 404 Not Found`
Search Flights
`GET /api/flights/search?departure=Bucuresti&arrival=Paris`
Raspuns: `200 OK | 404 Not Found`

3.5.7 Reservation Endpoints

Get All Reservations
`GET /api/reservations`
Raspuns: `200 OK`

Get Reservation by ID
`GET /api/reservations/{id}`
Raspuns: `200 OK | 404 Not Found`

Create Reservation
`POST /api/reservations`
Raspuns: `201 Created`

Update Reservation
`PUT /api/reservations/{id}`
Raspuns: `200 OK | 404 Not Found`

Delete Reservation
`DELETE /api/reservations/{id}`
Raspuns: `204 No Content | 404 Not Found`

Get Reservations by User
`GET /api/reservations/user/{userId}`
Raspuns: `200 OK`

Get Reservations by Flight
`GET /api/reservations/flight/{flightId}`
Raspuns: `200 OK`

Search Reservation
`GET /api/reservations/search?userId=2&flightId=5`
Raspuns: `200 OK | 404 Not Found`

3.6 Implementarea Frontend-ului

Frontend-ul aplicatiei este realizat folosind React si este structurat modular. Aplicatia comunica cu backend-ul prin API-uri REST si foloseste react-router-dom pentru routing. Fiecare componenta si pagina este conceputa pentru a fi reutilizabila si pentru a interactiona eficient cu starea aplicatiei si contextul de autentificare.

3.6.1 Structura generala

- api - contine functii pentru apelurile API catre backend: flightApi, reservationApi, userApi
- components - contine componente UI reutilizabile: FlightCard, Input, Button, NavBar
- context - contine contextul pentru autentificare: AuthContext
- hooks - contine hook-uri personalizate: useAuth pentru login, register si logout

- pages - contine pagini principale ale aplicatiei: HomePage, AllFlightsPage, ReservationsPage, EditFlightPage, AccountPage, LoginPage, RegisterPage
- types - contine definitii de tipuri TypeScript: Flight, Reservation, User, ReservationWithFlight

3.6.2 Routing-ul aplicatiei

Routing-ul este definit in App.tsx si permite navigarea intre pagini. Rutele principale sunt:

- /login - LoginPage
- /register - RegisterPage
- /home - HomePage
- /all-flights - AllFlightsPage
- /reservations - ReservationsPage
- /edit-flight - EditFlightPage
- /account - AccountPage

Navigarea se poate face si prin componenta NavBar, care afiseaza link-uri diferite in functie de rolul utilizatorului si starea de autentificare.

3.6.3 Contextul si autentificarea

- Contextul AuthContext pastreaza informatii despre utilizator si rolul acestuia.
- Variabila isAdmin indica daca utilizatorul curent este administrator.
- setIsAdmin permite actualizarea acestei valori.
- Hook-ul useAuth ofera metode:
  - login - autentifica utilizatorul folosind loginUser
  - register - inregistreaza utilizatorul folosind registerUser
  - logout - deconecteaza utilizatorul si reseteaza contextul
- Aceste metode interactioneaza cu backend-ul si actualizeaza starea aplicatiei si contextul.

3.6.4 Componente UI principale

- FlightCard
  - Afiseaza informatii despre un zbor: plecare, sosire, ora plecarii, durata, avion, pret
  - Permite rezervare, stergere zbor, editare zbor sau stergere rezervare in functie de rol si context
  - Date de intrare: obiect Flight, flag isReservation, reservationId, functia setReservations
  - Date de iesire: starea cardului modificata (rezervare creata sau stearsa, zbor sters)
- Input
  - Camp de input cu label
  - Date de intrare: label, tip input, valoare curenta, functia setValue
  - Date de iesire: actualizarea valorii in state-ul paginii
- Button
  - Buton reutilizabil care executa o functie la click
  - Date de intrare: text, handleClick, clasa CSS optionala
  - Date de iesire: apelarea functiei handleClick
- NavBar
  - Afiseaza meniul de navigatie si starea utilizatorului logat
  - Permite logout si navigarea catre cont sau alte pagini

3.6.5 Pagini si metodele lor

- LoginPage
  - Date de intrare: username, password
  - Metoda folosita: loginUser
  - Date de iesire: obiect utilizator (id, username, admin) sau mesaj de eroare
  - La succes: utilizatorul este salvat in sessionStorage si starea logged devine true
- RegisterPage
  - Date de intrare: username, password
  - Metoda folosita: registerUser
  - Date de iesire: obiect utilizator sau mesaj de eroare

  - La succes: redirectionare catre LoginPage cu mesaj de confirmare

- HomePage
  - Date de intrare: aeroport de plecare, aeroport de sosire
  - Metode folosite: getFlightByDepartureAndArrival, postReservation
  - Date de iesire: obiecte Flight sau lista de Flight
  - Utilizatorul poate cauta zboruri si rezerva direct din FlightCard
- AllFlightsPage
  - Date de intrare: none
  - Metode folosite: getAllFlights, deleteFlight
  - Date de iesire: lista de obiecte Flight
  - Administratorul poate edita sau sterge zborurile afisate
- ReservationsPage
  - Date de intrare: userId
  - Metode folosite: getReservationsByUserWithFlight, deleteReservation
  - Date de iesire: lista de ReservationWithFlight
  - Afiseaza rezervarile curente ale utilizatorului si permite stergerea lor
- EditFlightPage
  - Date de intrare: id-ul zborului, obiect Flight
  - Metoda folosita: updateFlight
  - Date de iesire: obiect Flight actualizat sau mesaj de eroare
  - Administratorul poate edita toate detaliile unui zbor
- AccountPage
  - Date de intrare: userId, username, isAdmin
  - Metode folosite: getUserById, updateUsername, deleteUser
  - Date de iesire: obiect utilizator actualizat sau confirmarea stergerii contului
  - Utilizatorul poate modifica username-ul sau sterge contul

3.6.6 Fluxul de date general

- Utilizatorul navigheaza intre pagini prin routing sau NavBar
- Pagini precum HomePage, AllFlightsPage si ReservationsPage apeleaza functii API pentru obtinerea datelor
- Pagini precum EditFlightPage si AccountPage permit modificarea datelor si transmit modificarile catre backend
- Componentele FlightCard, Input si Button faciliteaza interactiunea si vizualizarea datelor
- Contextul AuthContext mentine starea autentificarii si rolul utilizatorului pe intreaga aplicatie
- Backend-ul returneaza date JSON, care sunt afisate in UI sau folosite pentru actualizarea starii interne

4 Initializarea proiectului

4.1 Initializarea backend-ului

Pentru rularea backend-ului aplicatiei sunt necesare Java si Maven instalate pe sistem.

Pasii de initializare sunt urmatorii:

1. Clonarea repository-ului sau descarcarea codului sursa

   ```
   git clone <https://github.com/Adrian6303/CompanieZbor.git>
   cd CompanieZbor
   ```

2. Build-ul proiectului

   ```
   mvn clean install
   ```

3. Rularea aplicatiei

   ```
   mvn spring-boot:run
   ```

   Alternativ, aplicatia poate fi rulata folosind fisierul JAR generat:

   ```
   java -jar target/CompanieZbor-0.0.1-SNAPSHOT.jar
   ```

Dupa pornirea aplicatiei, serviciile sunt disponibile la:

- API Base URL: `http://localhost:8080/api`
- H2 Console: `http://localhost:8080/h2-console`

4.2 Initializarea frontend-ului

Pentru rularea aplicatiei frontend este necesara instalarea *Node.js* si a managerului de pachete *npm*.

Pasii de initializare sunt:

1. Navigarea in directorul aplicatiei frontend

   ```
   cd client
   ```

2. Instalarea dependintelor

   ```
   npm install
   ```

3. Pornirea aplicatiei

   ```
   npm run dev
   ```

Aplicatia frontend va fi accesibila in browser si va comunica cu backend-ul prin API-urile REST definite anterior.

5 Testarea backend-ului

Aplicatia foloseste doua categorii principale de teste: teste de integrare si teste unitare. Aceasta abordare permite verificarea corecta a comportamentului fiecarui strat al aplicatiei, dar si a modului in care componentele interactioneaza intre ele.

5.1 Arhitectura testelor

Testele sunt organizate pe aceeasi structura ca si aplicatia principala. Testele de integrare vizeaza stratul de controller si sunt rulate in context complet Spring Boot, verificand comunicarea dintre controller, service si repository, precum si raspunsurile HTTP returnate de API.

Testele unitare sunt dedicate stratului de service si verifica logica aplicatiei in mod izolat. In acest caz, accesul la baza de date este simulat prin intermediul Mockito, permitand testarea regulilor de business fara dependente externe.

5.2 Configurarea mediului de test

Pentru rularea testelor este utilizata o baza de date H2 in-memory, configurata special pentru mediul de test. Aceasta configuratie asigura izolarea completa intre rularile succesive si elimina problemele de blocare a fisierelor. Schema

bazei de date este generata automat la inceputul testelor si stearsa la final, iar fiecare test ruleaza intr-o tranzactie care este anulata dupa executie.

Aceasta abordare garanteaza faptul ca datele folosite in teste nu afecteaza datele reale ale aplicatiei si ca rezultatele obtinute sunt reproductibile.

5.3 Teste de integrare – Controller Layer

Pentru fiecare controller al aplicatiei sunt implementate teste de integrare care folosesc clasa MockMvc pentru a simula cereri HTTP fara a porni un server real. Aceste teste verifica functionarea corecta a endpoint-urilor REST, inclusiv procesarea cererilor, validarea datelor si construirea raspunsurilor.

Sunt testate atat scenariile de utilizare normala, cat si situatiile de eroare, cum ar fi accesarea unor resurse inexistente sau transmiterea unor date invalide. In acest mod se asigura respectarea standardelor REST si comportamentul robust al API-ului.

5.4 Teste unitare – Service Layer

Stratul de service este testat prin teste unitare care folosesc Mockito pentru simularea repository-urilor. Aceste teste verifica corectitudinea apelurilor catre repository-uri si logica aplicatiei, precum cautarea entitatilor, salvarea datelor si gestionarea relatiilor dintre utilizatori, zboruri si rezervari.

Testele sunt scrise astfel incat fiecare metoda sa fie verificata independent, folosind date controlate si rezultate asteptate clar definite.

5.5 Rularea testelor

Testele sunt rulate folosind Maven, fara a fi necesare configurari suplimentare. Pentru executarea tuturor testelor se foloseste comanda:

```
mvn test
```

Pentru rularea unei clase specifice de test se poate utiliza comanda:

```
mvn test -Dtest=UserControllerTest
```

Aceasta abordare permite testarea rapida a componentelor individuale in timpul dezvoltarii.

5.6 Acoperirea codului (JaCoCo)

Pentru masurarea acoperirii codului de catre teste este utilizat plugin-ul JaCoCo. Acesta genereaza rapoarte detaliate privind acoperirea liniilor de cod, a ramurilor si a metodelor testate.

Raportul de acoperire poate fi generat folosind comanda:

```
mvn test jacoco:report
```

Raportul rezultat este disponibil in directorul target/site/jacoco si poate fi analizat prin intermediul unui browser web. Nivelul de acoperire obtinut este ridicat, fiind testate toate endpoint-urile REST si toate metodele din stratul de service, inclusiv scenariile de succes si cele de eroare.

5.7 Bune practici utilizate

Testele respecta pattern-ul Arrange–Act–Assert, ceea ce asigura claritate si lizibilitate. Fiecare test este independent si nu depinde de rezultatele altor teste. Denumirile metodelor de test sunt descriptive si reflecta comportamentul verificat.

Utilizarea testelor automate, impreuna cu raportarea acoperirii prin JaCoCo, ofera o baza solida pentru mentenanta si extinderea aplicatiei

6 Feature-uri viitoare

Aplicatia este functionala si acopera toate cerintele de baza, insa poate fi extinsa pentru a oferi functionalitati avansate si o experienta imbunatatita utilizatorului:

- Preturi dinamice: preturile biletelor se vor ajusta automat in functie de cerere, perioada, locurile disponibile si alte criterii, oferind oferte actualizate in timp real.
- Alegerea locului in avion: utilizatorii vor putea selecta locul dorit in avion in momentul rezervarii, ceea ce creste confortul si controlul asupra calatoriei.
- Check-in online cu 24 de ore inainte: utilizatorii vor putea efectua check-in-ul online pentru zborurile rezervate, economisind timp si evitand aglomeratia la aeroport.
- Securitate mai avansata: se vor implementa masuri precum autentificare cu doi factori, criptarea datelor sensibile si protectii suplimentare impotriva atacurilor cibernetice.

7 Concluzie

Proiectul dezvoltat reprezinta o aplicatie completa pentru gestionarea zborurilor si a rezervarilor, cu un backend robust realizat in Java Spring si un frontend modern in React. Sistemul permite autentificarea utilizatorilor, cautarea zborurilor, realizarea rezervarilor si administrarea zborurilor de catre administratori.

Structura modulara, utilizarea componentelor reutilizabile si implementarea contextului de autentificare permit o dezvoltare ulterioara usoara si mentenanta eficienta. Testele unitare si de integrare asigura fiabilitatea aplicatiei si corectitudinea functionalitatilor.

Feature-urile viitoare vor aduce un nivel mai mare de interactivitate, personalizare si securitate, consolidand aplicatia ca o platforma profesionala pentru rezervarea biletelor de avion.